Implementation of traffic splitting using meter table in software-defined networking

Nattapong Kitsuwan, Eiji Oki

Department of Computer and Network Engineering, The University of Electro-Communications, Tokyo, Japan
E-mail: kitsuwan@uec.ac.jp

Published in *The Journal of Engineering*; Received on 20th July 2016; Accepted on 14th September 2016

Abstract: An implementation technique for traffic splitting using a meter table of a standard OpenFlow in a software-defined network is proposed. The proposed technique uses different differentiated services code point (DSCP) numbers, which are generally used to classify network traffic for quality of service (QoS) levels, in a packet header to decide an output port for a corresponding route. The meter table is generally designed for a QoS purpose. The OpenFlow standard provides two types of meter band, 'drop' and 'dscp_remark'. The proposed technique adopts the 'dscp_remark' type for a splitting traffic purpose, instead of the QoS purpose. The DSCP number of the packet is modified when the packet rate exceeds a pre-defined traffic rate. Packets with different DSCP numbers are sent out to the neighbour switches from their corresponding output ports. The original DSCP number is returned to the packets when they are sending out to the neighbour switches, so that the QoS is not affected.

1 Introduction

Efficient routing in Internet protocol (IP) networks increases network throughput and resource utilisation. Consequently, additional traffic can be inserted into the network. For an example, an approach that minimises the maximum link utilisation rate, called a congestion ratio, increases the admissible traffic, which enhances the routing performance. In this approach, the traffic is split and injected into multiple routes to reduce the congestion ratio.

To avoid network congestion, smart open shortest path first (S-OSPF), which is an extended version of the OSPF routing protocol [1], was presented. In S-OSPF, the traffic is split only at a source edge node. The packets are distributed to the neighbour nodes with the distribution ratios specified by the network controller. At the neighbour nodes, the packets are forwarded to the destination node based on the regular OSPF routing.

Software-defined networking (SDN) enables all network elements to be controlled by a central intelligent control and management platform, where an improved network programmability allows dynamic and flexible control of the routing elements. OpenFlow is a protocol that enables the control plane of the controller to interact with the data plane of the switches in the SDN network, so that the network can be adjusted. The forwarding instructions are based on flow entries, which are defined by a set of specific parameters.

Traffic splitting in the SDN network can be implemented using a group table, which is available from OpenFlow 1.1. The group table is able to represent additional methods of forwarding. With option group type '*select*', each packet is selected to be forwarded based on a selection algorithm, which is implemented in each switch. It is necessary to implement the selection algorithm on every switch in the network. The concept using the group table was implemented in [2]. In this implementation, the traffic is split into the primary and backup paths based on splitting policy, once traffic rate exceeds the pre-defined traffic rate.

Another implementation approach is splitting the traffic into multiple output ports based on a period of time [3]. The controller sends a flow configuration to adjust an output port of a flow entry every pre-assigned period of time. The flow messages from the controller are torrential in the network control plane. In [4], it takes 14 ms to send flow configurations from the controller until an OpenFlow switch finishes a flow installation. Accordingly, the frequency of sending flow configurations to modify the output port is restricted.

This paper proposes an implementation technique for traffic splitting by adopting a meter table, which is available from OpenFlow 1.3 [5]. It remarks packets with different differentiated services code point (DSCP) numbers and splits the traffic at a node based on the DSCP number. DSCP is a 6 bit field in the IP header, which identifies a level of service of a packet in the network. The meter table consists of several meter entries. Each meter entry includes a meter band, which specifies the rate of the band and the way to process the packet. The OpenFlow standard provides two types of meter bands, which are 'drop' and dscp_remark'. The option 'drop' is used to limit a packet rate. The meter band drops packets if the rate passing through the meter exceeds a pre-defined rate. The option 'dscp_remark' is used to define a simple DS (DiffServ) policer, as a quality of service (QoS). For example, DSCP = 0 and DSCP = 46 are used for best effort and high priority expedited forwarding, respectively. It is performed by modifying the DSCP field in the IP header of the packets whose rate exceeds the pre-defined rate. The proposed technique adopts the 'dscp_remark' option. A modification of DSCP number is used to remark the packets according to the pre-defined rate before the packets are sent out to desired output ports. Note that the output port returns the original DSCP number to the packets before sending them to a neighbour switch, so that the QoS of the traffic will not be changed.

2 OpenFlow meter table

The meter table consists of meter entries, defining per-flow meters. A meter measures an incoming packet rate and performs QoS operations including rate-limiting and DiffServ. A meter entry includes a meter band as one of the main elements. The meter band specifies a rate of the band and the way to process the packet. It includes a band type, a rate, and type specific argument. The band type defines how to process packets. The rate defines the lowest rate at which the band can apply. The type specific argument specifies the modified DSCP number. It is required if the band type is 'dscp_remark'.

3 Proposed implementation technique

The implementation technique adopts the 'dscp_remark' type of meter band from the standard OpenFlow to measure and to modify the DSCP number. Packets with different DSCP numbers are forwarded to desired output ports of the switch. The DSCP number is used as a label to divide the incoming packets into splits in the proposed technique, but it is unlikely used for the QoS purpose as in the

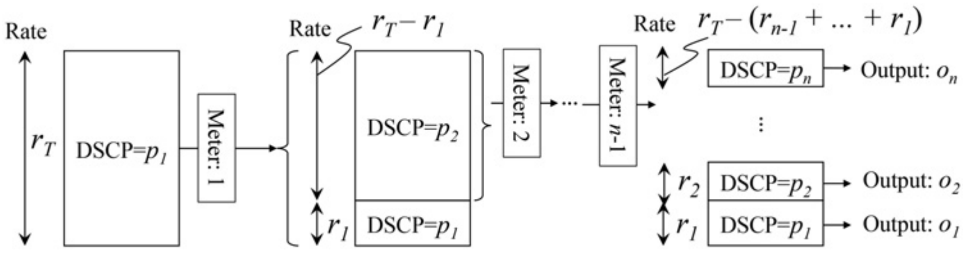


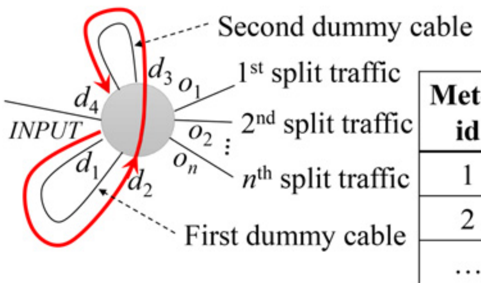
Fig. 1 *General concept of traffic splitting using meter table*

standard OpenFlow. Let n be a total number of splits. Let o_i be an output port for the ith split traffic, where $1 \le i \le n$. Let p_i be a DSCP number for the ith split traffic. Let r_i be a traffic rate of the ith split traffic. Let r_T be a total rate of incoming traffic, $r_T = \sum_1^n r_i$. Packets with p_i are sent to port o_i. The concept of the proposed technique is shown in Fig. 1. The incoming packets with traffic rate r_T and DSCP = p_1 arrive at the switch. Meter 1 measures r_T and divides the traffic into DSCP = p_1 with rate r_1 and DSCP = p_2 with the rest of rate. The traffic with DSCP = p_2 is measured by meter 2 and is divided into DSCP = p_2 with rate r_2 and DSCP = p_3 with the rest of rate. The process is repeated until the given number of splits is reached. After that, the packets with DSCP = p_i are sent out at output port o_i. The original DSCP numbers of the packets are returned when the packets are sent out at the output ports. Therefore, the QoS remains unchanged.

The architecture of each OpenFlow switch depends on switch vendors. For example, Pica8 switch [6] has a limitation of the multiple table architecture. The multiple tables are logically configured, but a ternary content-addressable memory of the switch physically supports only one table. The DSCP number of a packet cannot be modified without passing through an output port of the switch. To support all switch architectures, two dummy cables are introduced to recurve the packets via two output ports of the switch, in the case of three or more splits. Note that only one dummy cable is needed for the case of two splits.

Let us consider the case of $n \ge 2$ splits. The total number of necessary flow entries and meters for the splits are $2n - 1$ and $n - 1$. Fig. 2*a* shows the definition of ports. Let d_1 and d_2 be an output port to and an input port from the first dummy cable, respectively. Let d_3 and d_4 be an output port to and an input port from the second dummy cable, respectively. The process of traffic splitting is as follows:

- *Step 1:* Set $i = 1$. The DSCP number of the source traffic is defined by p_i.
- *Step 2:* Packets are received from port INPUT.
- *Step 3:* Rate of packets with the DSCP number p_i is measured by meter i. The DSCP number of packets whose rate exceeds r_i is defined by p_{i+1}. DSCP number of other packets remains p_i.
- *Step 4:* If i is an odd number, packets are sent out at port d_1. Port d_1 modifies the DSCP number of packets (if needed) as defined in step 3. Otherwise, packets are sent out at port d_3. Port d_3 modifies the DSCP number of packets (if needed) as defined in step 3.
- *Step 5:* d_2 and d_4 receive packets returned from d_1 and d_3, respectively.
- *Step 6:* Packets with DSCP number p_i are sent out at port o_i.
- *Step 7:* Increase i by one.
- *Step 8:* If $i < n$, repeat from step 3. Otherwise, go to step 9.
- *Step 9:* Packets with DSCP number p_{i+1} are sent out at port o_{i+1}.



a

Meter table

Meter id	Band type	Rate	Type specific argument
1	dscp_remark	r_1	prec_level=p_2
2	dscp_remark	r_2	prec_level=p_3
…	…	…	…
n-1	dscp_remark	r_{n-1}	prec_level=p_n

Flow table

x	Match field	Actions
1	ip,in_port=*INPUT*	meter:1,output:d_1
$2n$ - 1	ip,in_port=$d_{[(x+2) \text{ MOD } 4]+1}$, ip_dscp=$p_{(x+1)/2}$	output:$o_{(x+1)/2}$
Odd number	ip,in_port=$d_{[(x+2) \text{ MOD } 4]+1}$, ip_dscp=$p_{(x+1)/2}$	meter:$(x+1)/2$, output:$d_{[(x+4) \text{ MOD } 4]}$
Even number	ip,in_port=$d_{[(x+2) \text{ MOD } 4]+2}$, ip_dscp=$p_{x/2}$	output:$o_{x/2}$

b

Fig. 2 *Pattern of meter and flow entries*
a Port definition
b Meter and flow entries pattern

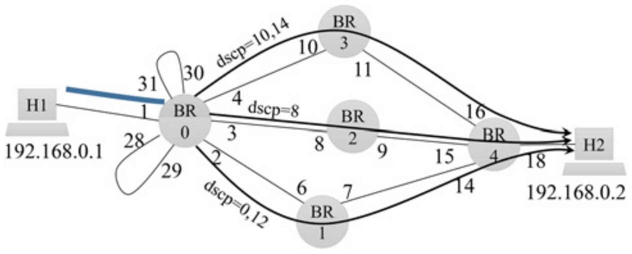
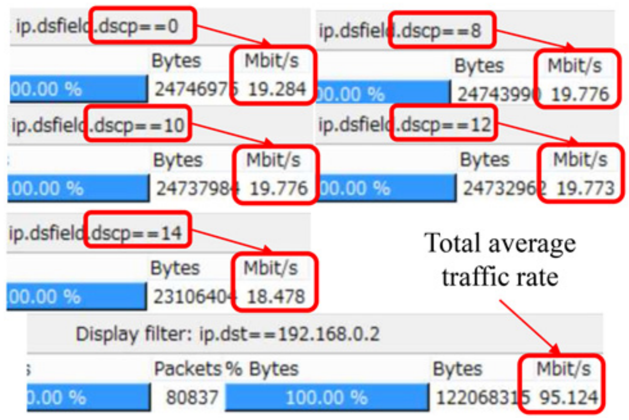


Fig. 3 *Network topology*

Fig. 2*b* shows a general pattern of flow entries of the switch in which the traffic is split based on the above process. Let x be an index of flow entry. x is used as a parameter to calculate the pattern. Only match field and actions are contained in the flow entry. At the initial state, r_i is configured by meter i.

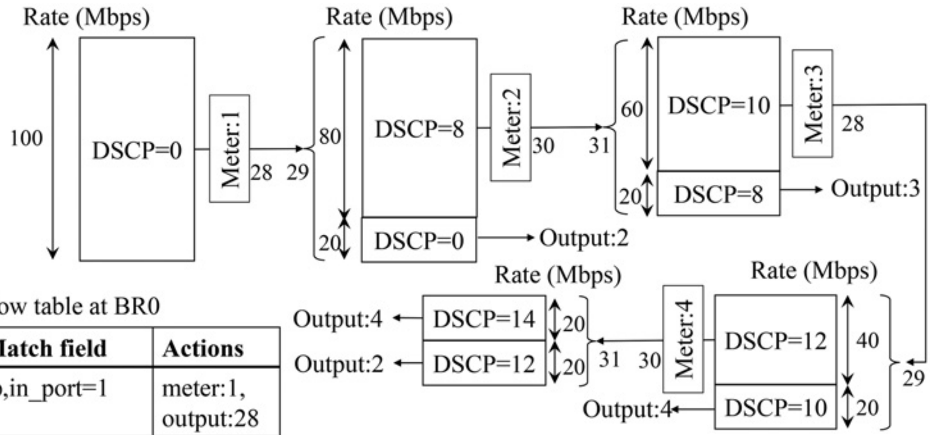
4 Implementation

The idea of the proposed implementation technique is confirmed by an experiment using a Pica8 switch. Five bridges, from BR0 to BR4, are created as virtual switches. Fig. 3 shows network topology and port number. BR0 modifies the DSCP number of packets when the traffic rate exceeds the defined rate. Ports 28 and 29 are used for the first dummy cable, $d_1 = 28$ and $d_2 = 29$. Ports 30 and 31 are used for the second dummy cable, $d_3 = 30$ and $d_4 = 31$. BR0 also distributes packets to desired output ports based on the DSCP number. BR1–BR4 just forward every packet to their output port. In the experiment, we use a testing tool, called iPerf3 [7], to create a transmission control protocol traffic with 100 Mbps from H1 to H2. Traffic rate of 100 Mbps is equally split into five parts. The first to the third parts are forwarded to ports 2, 3, and 4, respectively. The fourth and the fifth parts are forwarded



```
ip.dsfield.dscp==0              ip.dsfield.dscp==8
          Bytes     Mbit/s                Bytes     Mbit/s
00.00 %   24746975  19.284      00.00 %   24743990  19.776
ip.dsfield.dscp==10             ip.dsfield.dscp==12
          Bytes     Mbit/s                Bytes     Mbit/s
00.00 %   24737984  19.776      00.00 %   24732962  19.773
ip.dsfield.dscp==14
          Bytes     Mbit/s             Total average
00.00 %   23106404  18.478             traffic rate
     Display filter: ip.dst==192.168.0.2
          Packets  % Bytes          Bytes      Mbit/s
0.00 %    80837    100.00 %         122068315  95.124
```

Fig. 5 *Traffic rate for each DSCP number*



a

Flow table at BR0

Match field	Actions
ip,in_port=1	meter:1, output:28
ip,in_port=29, ip_dscp=0	output:2
ip,in_port=29, ip_dscp=8	meter:2, output:30
ip,in_port=31, ip_dscp=8	output:3
ip,in_port=31, ip_dscp=10	meter:3, output:28
ip,in_port=29, ip_dscp=10	output:4
ip,in_port=29, ip_dscp=12	meter:4, output:30
ip,in_port=31, ip_dscp=12	output:2
ip,in_port=31, ip_dscp=14	output:4

d_1=28	p_1=0	o_1=2
d_2=29	p_2=8	o_2=3
d_3=30	p_3=10	o_3=4
d_4=31	p_4=12	o_4=2
	p_5=14	o_5=4

Meter table at BR0

Meter id	Band type	Rate	Type specific argument
1	dscp_remark	20Mbps	prec_level=8
2	dscp_remark	20Mbps	prec_level=10
3	dscp_remark	20Mbps	prec_level=12
4	dscp_remark	20Mbps	prec_level=14

b

Fig. 4 *Process of DSCP modification*
a Paradigm of implementation
b Flow and meter entries at BR0

to ports 2 and 4, respectively. Returning the original DSCP number to the packets is omitted since all packets with different DSCP numbers will be confirmed at the destination.

The process of DSCP modification is shown in Fig. 4. Fig. 4*a* illustrates a paradigm of the implementation. The incoming traffic has 100 Mbps with DSCP = 0. Packet rate of the incoming traffic is measured by meter 1.

DSCP = 8 is defined for packets with traffic rate over 20 Mbps. All packets are sent to the output port 28. DSCP number of packets is modified. The traffic is then split into two parts, which are 20 Mbps of packets with DSCP = 0 and 80 Mbps of packets with DSCP = 8. After the packets recurve back to port 29, packets with DSCP = 0 are directly forwarded to output port 2. Meter 2 measures the traffic of packets with DSCP = 8. DSCP = 10 is defined for packets with traffic rate over 20 Mbps. The DSCP number of packets with defined DSCP = 10 is modified at port 30. All packets are sent out at port 30 and recurved back to port 31. Packets with DSCP = 8 are then sent to output port 3. Meter 3 measures the traffic of packets with DSCP = 10. DSCP = 12 is defined for packets with traffic rate over 20 Mbps. The DSCP number of packets with defined DSCP = 12 is modified at port 28. All packets are sent out at port 28 and recurved back to port 29. Packets with DSCP = 10 are then sent to output port 4. Meter 4 measures the traffic of packets with DSCP = 12. The DSCP number of packets with defined DSCP = 14 is modified at port 30. All packets are sent out at port 30 and recurved back to port 31. Finally, packets with DSCP = 12 and DSCP = 14 are sent to output ports 2 and 4, respectively. Fig. 4 shows meter and flow entries in switch BR0 to corresponding with the above scenario. They are calculated as the definition in Fig. 2. Note that returning the original DSCP number is not configured, since we want to measure the traffic rate of each split traffic at H2.

All packets are captured at H2 using Wireshark. The average received traffic rate of the total packets at H2 is 96.7 Mbps. The average traffic rates of packets with DSCP = 0, 8, 10, 12, and 14 are 19.284, 19.776, 19.776, 19.883, and 18.478 Mbps, as shown in Fig. 5. We also check the byte counters at BR1–BR3 to confirm that the packets with different DSCP numbers are forwarded to their corresponding paths. This result confirms the concept of using meter table to modify the DSCP number for traffic splitting.

5 Conclusion

An implementation technique for traffic splitting using a meter table in an SDN network was proposed. In the proposed technique, a DSCP number in IP packet header is modified based on pre-defined traffic rates. After modification of the DSCP number, packets are sent out at corresponding output ports of the switch, each output port is associated with a corresponding DSCP number. The result of the implementation confirms the idea of the proposed technique. The traffic is freely split as a user requirement.

6 References

[1] Mishra A.K., Sahoo A.: 'S-OSPF: a traffic engineering solution for OSPF based on best effort networks'. Proc. IEEE Global Communications Conf., Washington, DC, USA, November 2007, pp. 1845–1849

[2] Braun W., Menth M.: 'Load-dependent flow splitting for traffic engineering in resilient OpenFlow networks'. Proc. Int. Conf. and Works Networked Systmes (NetSys), Cottbus, German, March 2015, pp. 1–5

[3] Oki E., Nakahodo Y., Naito T., *et al.*: 'Implementing traffic distribution function of smart OSPF in software-defined networking'. Proc. 21st Asia-Pacific Conf. Communications (APCC2015), Kyoto, Japan, October 2015, pp. 239–243

[4] Kitsuwan N., McGettrick S., Slyne F., *et al.*: 'Independent transient plane design for protection in OpenFlow-based networks', *IEEE/OSA J. Opt. Commun. Netw.*, 2015, **7**, (4), pp. 264–275, doi: 10.1364/JOCN.7.000264

[5] 'Open Networking Foundation – OpenFlow v1.3', 2012

[6] http://www.pica8.com/, accessed January 2016

[7] http://www.software.es.net/iperf/, accessed January 2016